Multi-Resolution Point-Sample Raytracing

Michael Wand Wolfgang Straßer
WSI/GRIS, University of Tübingen

Abstract

We propose a new strategy for raytracing complex scenes without aliasing artifacts. The algorithm intersects anisotropic ray cones with prefiltered surface sample points from a multi-resolution point hierarchy. The algorithm can be extended to capture effects of distributed raytracing [7] such as blurry reflections, depth of field, or soft shadows. In contrast to former anti-aliasing techniques based on cone tracing, the multi-resolution algorithm can be applied efficiently to scenes of high complexity. The running time does not depend on the variance in the image as this is the case for the prevalent stochastic raytracing techniques. Thus, the new technique is faster than stochastic raytracing for images with many high frequency details.

Keywords: *Point Sample Rendering, Raytracing, Multi-Resolution Rendering, Anti-Aliasing.*

1 Introduction

Raytracing [5, 29] is a very general image generation technique that is able to simulate many global lighting phenomena such as shadows or reflections. The algorithm traces a ray into the scene through each pixel of the image. This sampling approach is prone to aliasing artifacts. The common solution to this problem is stochastic supersampling: Multiple rays are shot through each pixel and the results are averaged using a suitable filter kernel. The central limit theorem guarantees that this average will converge to the integral over the pixel (weighted by the filter kernel) stochastically. The problem of this approach is slow convergence: In the worst case, the convergence rate (standard deviation of the error) is $O(\sigma/\sqrt{n})$ for n sample rays and a standard deviation σ of the color estimator [18]. In regions of high variance, a large sample size of up to some hundred rays per pixel is needed to avoid noise artifacts.

In some cases, the convergence rate can be improved by stratification or importance sampling [9]. Importance sampling requires a priori knowledge and therefore, it is often not applicable. In cases of complex, irregularly structured image signals, stratification also does not lead to a better convergence rate: Sample sets from a (in the worst case) purely random image signal will have the same probability distribution for any set of sampling positions [18]. This argument applies to quasi-random samples (such as samples from a Halton sequence [15]) as well.

An alternative are methods that trace extended ray volumes such as cone tracing or beam tracing [3, 10, 11, 16, 25]. These methods do not shoot infinitesimally small rays into the scene but larger cones with a cross-section corresponding to a pixel in the image. These techniques render anti-aliased images using only one ray per pixel. However, they suffer from a different kind of complexity problem: In a highly detailed scene, the cross-section of a ray cone may intersect with an arbitrarily large set of primitives. Thus, the intersection computations become prohibitively expensive. For this reason, methods following this paradigm are usually only applied to models of low complexity.

A possible solution is to use a hybrid approach [4, 8]: Extended ray cones are used to detect boundaries in objects space. Then super-sampling is used to integrate over the cross-section of the ray. These techniques allow a good control of the sampling density used for oversampling. Nevertheless, in regions of high variance, they suffer from the same convergence problems as the purely stochastic methods.

In this paper, we propose a new cone-tracing algorithm that uses a multi-resolution point sample hierarchy to speed up the intersection calculations: Instead of intersecting the extended ray volume with potentially millions of geometric primitives we use only a few sample points with a spacing matching the ray footprint. The sample points store precomputed average color attributes and differential properties such as an average normal and curvature information [14, 21]. The ray footprints are estimated using a ray model similar to ray differentials [12] for anisotropic antialiasing.

We compare a prototype implementation of our multi-resolution point hierarchy raytracer with a conventional and a distributed raytracer based on the same code basis. Our experiments show that the new technique produces antialiased images at about 4 times the costs of the simple, non-antialiased raytracer, independent of the variance of the image. Approximations of effects like soft shadows and blurry reflections can be

obtained by modifying the ray footprint, at no additional costs. In images with high variance areas such as soft-shadow areas or models with high frequency texture, the new technique is much faster than the distributed raytracer and guarantees images without noise artifacts.

2 Related Work

Cone/beam-tracing techniques: Cone tracing [3, 16] calculates the interaction of ray cones through each pixel with the scene geometry. The technique is able to render antialiased images and to approximate effects such as soft shadows and blurry reflections. Beam tracing [10, 11] starts with the view frustum as initial beam and successively clips it to polygonal objects to generate sub-beams. Shinya et al. discuss models for the interaction of "pencils" of rays with reflecting and refracting surfaces [25]. A more general and robust model of the interactions between beams of light and surfaces is described by Igehy [12]: The first order derivatives of the ray properties (origin, direction) with respect to the screen coordinate system are considered to estimate ray footprints. The chain rule is used to deal with multiple surface interactions. The technique is applied to estimate footprints for prefiltered texture mapping. Schilling [24] uses similar techniques to perform antialiasing of environment maps.

Our new algorithm is based on the conceptual framework of cone tracing. We use a point based multi-resolution hierarchy to overcome the complexity problems of this approach: When large cones have to be intersected with complex geometry, the multi-resolution hierarchy allows adapting the model resolution to a coarser level to bound the computational efforts. Additionally, we use a more flexible anisotropic ray model than the original cone tracing algorithm based on some ideas of the ray differentials model [12] and Schilling's technique [24].

Stochastic raytracing: Distributed raytracing [7] is a standard technique for antialiased raytracing and the simulation of higher dimensional sampling effects such as soft shadows from extended light sources or blurry reflections: The intensity of every pixel is calculated as a weighted average of irregularly distributed sample rays. The technique can be implemented efficiently using adaptive supersampling by either examining image variance [19, 20] or object space features [4, 8]. The latter approach leads to a hybrid cone/beam tracing and supersampling algorithm.

Our algorithm does not try to perform a complete global illumination simulation that is possible with advanced stochastic raytracing techniques such as photon tracing or path tracing [13] but is limited to some special effects similar to those described by Cook et al. [7]. In comparison with Cook's "distributed raytracing" methods, our algorithm is only able to compute an approximate solution while distributed raytracing converges stochastically to the exact solution. The advantage of our method is that it produces plausible images without noise artifacts. This is sometimes more important than physical correctness, e.g. for rendering animations.

Point sample rendering: Multi-resolution point representations store clouds of surface sample points in a spatial hierarchy to approximate geometric objects at different levels of detail [21, 22, 27]. The attributes of the points can be prefiltered to allow an image reconstruction without aliasing during rendering [21, 30]. As every point can store only a fixed set of information, a point sample can only approximate the original geometry it represents. Different models of representation are possible: We use differential point samples [14] which include curvature information that is needed to antialias secondary rays [12]. We use a technique similar to surface splatting [30] to reconstruct pieces of surface for ray-point and ray-triangle intersections.

Raytracing of point based representations has already been explored by different authors: Lischinski et al. and Agrawala et al. [2, 17] describe raytracing techniques for layered depth images using hierarchical acceleration data structures. Our ray-point cloud intersection algorithm is similar to of that of Schaufler et al. [23]. Adamson et al. [1] point out that this technique does not necessarily lead to a uniquely defined surface between the sample points and propose an enhanced method based on a local projection operator. For our algorithm, we do not need a unique surface reconstruction: Because of the multi-resolution data structure, the point cloud is never seen under magnification. For close-ups, the algorithm automatically uses the original primitives of the geometric model [28].

The remainder of the paper is structured as follows: In the next section, we describe the multi-resolution point hierarchy. Section 4 describes the ray model and the intersection calculations with extended rays. Section 5 puts all components together and describes the multi-resolution ray tracing algorithm. Afterwards, we present results obtained with our implementation of the algorithm and conclude with some ideas for future research.

3 The Multi-Resolution Hierarchy

The input to our algorithm is a set of triangles: Each triangle is tagged with a set of material properties (color, reflectivity, transparency etc...) and a normal for each of its vertices.

The multi-resolution hierarchy consists of point samples that approximate pieces of the surface of the objects. Besides its position, each point stores average material and differential surface properties [14]: The

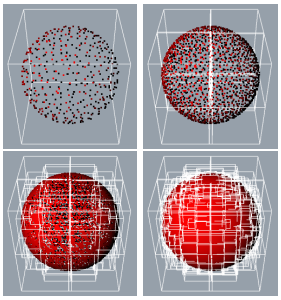


Figure 1: The point hierarchy

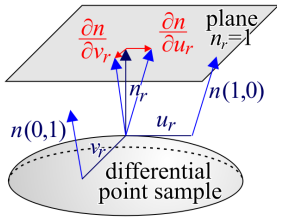


Figure 2: Differential point samples

material properties are an average color, reflectivity, transparency, Phong parameters, surface roughness, and index of refraction. The differential surface properties are the average normal and the derivatives of the normal in two orthogonal tangential directions (Figure 2). This second order surface approximation will be used by the raytracer later on to calculate the shape of the secondary rays.

3.1 Building the Hierarchy

We build the multi-resolution point hierarchy in a preprocessing step using a technique similar to that described by Wand et al. [28] which is a variant of the surfels data structure [21]. We start preprocessing by calculating a bounding cube for the triangle set. Then we choose a set of representative sample points at a fixed resolution (the point spacing is proportional to the side length of the box) and estimate average surface properties for them. If a triangle obtains more than only a few (say more than 3) representative points, it is stored in the current octree box and excluded from further point sampling. All other triangles are sorted into the child nodes and the algorithm is continued recursively for the child nodes until all triangles have reached their sampling limit. After that, we have a spatial hierarchy with triangles in the leaf nodes and approximating point clouds along with some large triangles in the inner nodes. The root of our hierarchy provides a very rough approximation of the scene and the resolution is increased when we go down in the hierarchy until we end up at the original triangles which represent the highest resolution available (Figure 1).

3.2 Calculating Point Samples

How do we generate the point clouds that approximate the geometry in the inner nodes? Our sampling strategy is divided conceptually into two steps: Firstly, the generation of *representative* points and secondly, the estimation of their *average surface properties*.

To choose representative points, we fix a three dimensional quantization grid of k^3 voxel for the current octree box. Then we draw a set of random sample points that are uniformly distributed on the triangles. For each voxel that contains at least one sample point, a representative point is created in the center of the voxel. As analyzed in [28], a relatively small number of random samples are sufficient to guarantee that the surface is covered with representatives with a maximum distance of at least the diagonal voxel distance.

In order to estimate the average surface properties, we increase the minimum sampling density by an oversampling factor of c (typically $c = 100$). The additional sample points are then used to compute prefiltered surface properties during preprocessing[1]: Each representative point is assigned a Gaussian weighting function (with a standard deviation of one voxel side length). The material properties (color, transparency etc...) as well as the average normal of the representative point are calculated as weighted average of the properties of the neighboring points in the sample set.

To obtain the derivatives of the normal, we fix an (arbitrary) tangential coordinate system (u_p, v_p) orthogonal to the average normal of the representative point and project all neighboring points into this coordinate system yielding points (u_i,v_i). For each point, we calculate a two dimensional normal deviation $\Delta n(u_i,v_i)$: The two components are the deviation to the original normal in u_p and v_p coordinates for a fixed third component n_p=1 in normal direction (see Figure 2). This means, we measure the normal deviation as projection in a plane parallel to the tangential plane with distance one [24].

To describe the surface curvature, we fit a bilinear function $n(u,v) = n(0,0) + u\cdot\partial n/\partial u + v\cdot\partial n/\partial v$ to the normal deviations $\Delta n(u_i,v_i)$ by solving a weighted least square problem. The derivatives of this function $(\partial n/\partial u, \partial n/\partial v) = \nabla n$ are stored in the representative point along with the orientation of the tangential coordinate system. Note that we fit a function to the normal vectors rather than fitting a height field to the positions of the sample points. This is necessary because we are using normal interpolated triangles as input. Thus, the spatial deviation of the sample points might not match the specified normals. After that, all point properties are quantized to small integer values (8 bit for material properties and position, 16 bit for differential properties) so that they can be stored compactly in the hierarchy.

[1] Because of the random sampling, we need a large sample set to avoid noise artifacts. This is a similar problem as in distributed raytracing. However, the sampling is done during preprocessing and in object space. Sampling in object space is much more efficient than sampling in the image space where an expensive inverse ray intersection problem must be solved for every sample. Thus, this is no severe efficiency problem. Even for larger models, typical preprocessing times are in the range of a few minutes (see Table 1).

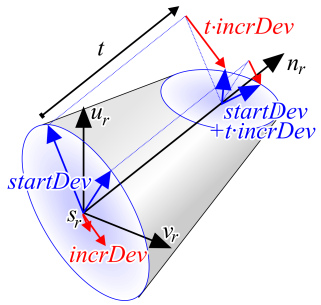


Figure 3: Ray coordinates

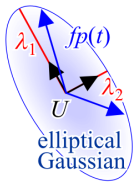


Figure 4: Principal component analysis

4 Representing Ray Cones

This section describes our ray cones model that is used to estimate the ray footprint. Generally, ray footprints are obtained by approximating the propagation of the set of all rays through a single pixel through the scene. For primary rays, this is easy. Dealing with secondary rays is more involved: There are two different basic approaches: First, one can estimate the ray footprint by considering the differences between adjacent rays in image space [8]. This approach leads to problems at the boundary of objects, where special processing is necessary. The second opportunity is to use differential information at the point of intersection, i.e. derivatives of the normal, to estimate the broadening or focusing effect of the surface on the incoming ray. We use the second strategy in a similar way as Ighegy [12]. This approach fits especially well in the context of a multiresolution renderer where differential properties can be precomputed for different levels of resolution.

4.1 Footprint Estimation

Our ray model is based on a linear approximation: Every ray r defines a local coordinate system, consisting of its origin s_r, its normalized direction n_r, and two tangential directions u_r, v_r (see Figure 3). The footprint is now described in the local u_r and v_r coordinates: Each ray stores two 2×2 matrices *startDev* and *incrDev*. The columns of *startDev* contain two vectors (in u_r,v_r coordinates) defining the footprint coordinate system at the ray parameter $t = 0$. The footprint at larger values of t is defined as

$$fp(t) = startDev + t \cdot incrDev. \qquad (1)$$

The footprint coordinate system is used to associate a filter kernel with each ray parameter t. We use an elliptical Gaussian filter [30] defined by:

$$weight(u,v) = e^{-\left(fp(t)^{-1}\begin{pmatrix} u \\ v \end{pmatrix}\right)^2} = e^{-(u,v)\,fp(t)^{-2}\begin{pmatrix} u \\ v \end{pmatrix}} \qquad (2)$$

Thus, the exponential decay of the filter weight is given by a quadric form of the coordinates of a point in local ray coordinates (u,v) using the matrix $fp(t)^{-2}$. This matrix is real and symmetric. Therefore, it can be decomposed into an eigensystem representation

$$fp(t)^{-2} = U^T \begin{pmatrix} \lambda_1^{\,2} & 0 \\ 0 & \lambda_2^{\,2} \end{pmatrix} U \qquad (3)$$

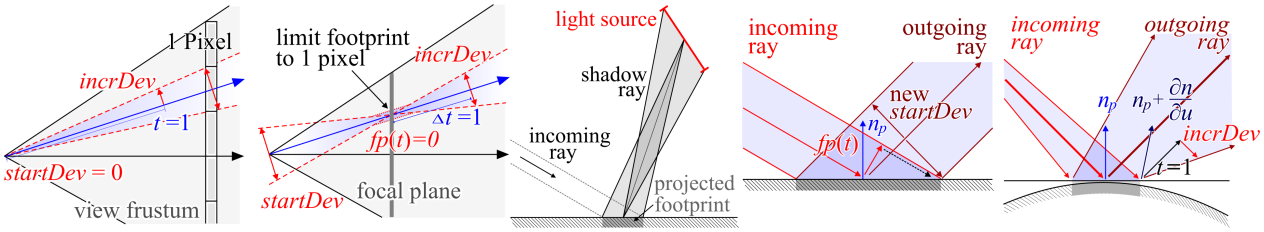
with an orthogonal Matrix U (see Figure 4). We will use this decomposition for different purposes, e.g. to enhance the numeric stability of triangle intersection calculations or to estimate the minimum diameter of the ray footprint.

4.2 Ray - Surface Interaction

In the following, we discuss how we obtain ray parameters for primary and secondary rays as well as for shadow rays:

Primary Rays: Primary rays are constructed by specifying zero deviation at the ray origin and an increment matrix that broadens the ray so that it matches the extents of the pixels in the image plane (Figure 5a). This also allows to model depth of field effects: The lens model used by Cook [7] leads to a diameter of the ray footprint that increases linearly to both sides of the focal plane (Figure 5b). This effect can be modeled by setting values for the deviation matrices such that the footprint coordinates are zero in the focal plane. To avoid aliasing in the focal plane, we must additionally compare the ray diameter with that of a conventional primary ray and take the maximum of both.

Shadow Rays: We allow arbitrary ellipses in three-space as light sources. The footprint of the shadow ray consists of two parts: Firstly, an elliptical cone with the cross-section of the light source at the source and zero at the current surface intersection point, and secondly, an elliptical cone with the cross-section of the footprint of the intersection at the surface and zero diameter at the light source (Figure 5c). The footprint at the surface intersection point is calculated by projecting the incoming ray footprint onto the tangent plane of the surface intersection point. The resulting footprint coordinate vectors are then transformed into ray coordinates. The footprint at the light source is obtained by transforming the two axes of the light source ellipse in ray coordinates. We then form two ray cones according to our linear model (Equation 1). The footprint at any ray parameter t is given by the convolution of the two footprint matrices of the two rays. The convolution can be computed by simply adding the squared footprint matrices [30] before the evaluation of Equation 2. Another possibility is to use only one ray and interpolate between the start and the end coordinate system by matching (also by possibly mirroring) the footprint coordinate axes with similar direction. This is less exact because the linear interpolation cannot capture a potentially rotating, asymmetric footprint without distortion. How-



a) primary rays b) depth of field c) shadow rays d) footprint projection e) reflected rays

Figure 5: Ray parameter setup for primary and secondary rays

ever, the results in practice are satisfactory.

Reflected Rays: We first reflect the center ray of the incoming ray cone at the normal of the intersection point. This yields a center ray for the reflected ray cone. It remains to calculate the ray footprint parameters *startDev* and *incrDev* of the reflected ray cone. To calculate *startDev*, we project the footprint of the incoming ray in ray direction onto the local tangent plane of the intersection point p (Figure 5d). This yields two vectors u_{fp}, v_{fp} that describe a footprint ellipse on the surface. Then we express the two vectors u_{fp}, v_{fp} in local ray coordinates u_r, v_r of the outgoing ray to obtain the columns of *startDev*.

To calculate *incrDev*, we estimate the normal directions at the points $p+u_{fp}$, $p+v_{fp}$ using the first order approximation matrix ∇n for the normals at the point of intersection. For each of the two points, we compute a direction of reflection, as shown in Figure 5e. Then the differences between the central direction of reflection and the directions at the two points are expressed in the local ray coordinates u_r, v_r to obtain the columns of *incrDev*.

Transmitted / Refracted Rays: Transmitted rays are handled the same way as reflected rays. The only difference is that the incoming vectors are refracted instead of reflected to calculate the outgoing directions. A small problem arises with total reflection: It is possible that a part of the rays in the footprint is reflected while others are refracted. Currently, we just decide for refraction or total reflection based on the direction of the center ray. However, this can lead to aliasing at the border of reflection and refraction. In such cases, a better opportunity would be to send two ray cones and blend together the results of the two rays.

4.3 Ray-Point Intersection

To compute the intersection between a surface sample point p and an extended ray r, we first express the point in ray coordinates: Let d be the difference vector between the point p and the ray origin. The scalar products between d and the (orthogonal) ray coordinates n_r, u_r and v_r (Figure 3) express the point coordinates in ray coordinates. The n_r component is the ray parameter t at the point of the ray that is closest to the sample point. Thus, we can evaluate $fp(t)$ (Equation 1) and calculate a weight for the point p (Equation 2).

4.4 Compositing

If a ray intersects with a piece of surface, there are usually many sample points with a non-neglectable weight[2] that contribute to a local surface reconstruction. We calculate a weighted average of all point samples using the weights obtained from the footprint matrix. This means, we use the Gaussian ellipse as reconstruction filter for a prefiltered, point sampled signal given by the points in the hierarchy [30]. This leads to effective antialiasing because the extents of the Gaussian ellipse correspond to the spacing of the pixels in the image plane.

The averaging is done by adding together the attributes of the points multiplied with their weight and then dividing by the weight sum. Special care must be taken with the differential properties: The matrices with the two derivative vectors of the normal in u_p and v_p direction cannot be averaged directly because the reference directions (u_p, v_p) vary with each point. Therefore, a coordinate system transformation from the coordinate system of the point to a common coordinate system must be performed. A cheaper alternative is to neglect aliasing of the surface derivatives as it would not lead to visible artifacts for most scenes and to use nearest neighbor sampling for the differential properties.

A second issue is occlusion: We must exclude all occluded points from the reconstruction. Furthermore, the edges of the objects should be blended with the background for silhouette antialiasing. To deal with this problem, we adopted again the strategy proposed by Zwicker et al. [30]: Each ray stores a simplified one pixel A-buffer. The buffer consists of a list of surface fragments, sorted in depth order. Each fragment stores all point attributes, its current sum of weights and a

[2] For efficiency reasons, the Gaussian filter is clipped to zero for all points that would yield a small weight (although it has theoretically infinite support). We clip at a weight below 2% (≙ 2×standard deviation as argument).

depth interval. When we encounter a new point-ray intersection, we search the list for the matching depth: Each point is initially assigned a depth tolerance interval proportional to the point spacing in the hierarchy node it has been taken from. If this interval overlaps with an interval in the list, the point is merged with the fragment (i.e. the attributes and weights are added) and the depth intervals are united. If the point does not overlap with a fragment in the list, it is inserted as a new fragment. The depth interval of the new fragment is set to its depth tolerance interval corresponding to its point spacing.

After all points have been inserted into the list and probably merged to larger fragments, the remaining fragments are blended together by alpha-blending: The point hierarchy is constructed to guarantee that any fragment in the interior of any surface has at least a weight of one. Therefore, we can interpret weight sums of less than one as alpha values to blend edges. Early ray termination is achieved by tracking the resulting alpha value for the whole ray at each ray intersection event and terminating the ray when a value of one is reached.

4.5 Ray-Triangle Intersection

Our multi-resolution hierarchy uses the original triangles to represent the highest resolution of the model. Thus, we must also compute anti-aliased intersections of extended rays and triangles.

This is done in three steps: Firstly, we transform the vertices of the triangle in ray coordinates. Then we evaluate the footprint matrices at the n_r-components. These matrices define a coordinate system in which the reconstruction filter is just a unit Gaussian. Therefore, in the second step, we transform the coordinates of the three vertices of the triangle in the footprint coordinate systems at the three points. We end up with a two dimensional triangle and a unit Gaussian around the center of the coordinate system. In the last step, we calculate the distance d of the closest point of the triangle to the origin and set the weight of the intersection to $\exp(-(d-0.5)^2)$. This leads to triangles with a boarder blurred by the ray footprint. The interpolation of the footprint along the edges of the triangle is done implicitly as we use three different matrices for the transformation of the three vertices. The ray intersections are then treated like point samples in the compositing step.

4.6 Ray-Bounding Volume Intersection

To perform the hierarchy traversal, we also need to compute intersections of extended rays with bounding volumes. A conservative intersection test can be performed using similar techniques as for the intersection with triangles: The vertices of the bounding volume (e.g. axis aligned bounding boxes) are transformed into ray-footprint coordinates. We obtain a 2-dimensional polygon and we must determine whether it intersects a circle around the origin, representing the drop-off radius of the Gaussian filter.

However, this test is quite expensive (e.g. 8 transformations for bounding boxes). Thus, we use a cheaper test in the current implementation: A bounding sphere is used as bounding volume. The center of the sphere is projected into the orthogonal unit coordinates of the filter (the main axes of the ellipse, matrix U in Equation 3). The eigenvalues of the filter coordinates form an axis aligned rectangle around the origin. The rectangle is tested for intersection with the projected bounding sphere, which is approximated by an axis aligned square, too. For the performance of the algorithm, it is very important to do the intersection test with elliptical ray cones (no circular approximation). Otherwise, queries with highly anisotropic ray cones become very expensive. The simple test accounts for this using the main axis transformation.

5 Multi-Resolution Raytracing

5.1 The Algorithm

Now we can put all things together and describe the complete multi-resolution raytracing algorithm. The algorithm starts at the root bounding box and then recursively performs the following steps:

1. intersection test *current bounding box* ⇔ *ray*
2. **if** *no intersection* **then return**
3. calculate *ray* ⇔ *triangle* intersections
4. compare *min. ray footprint* with *sample spacing*
5. **if** *point samples are dense enough* **then**
   calculate *ray* ⇔ *point* intersections
   else
   recursively traverse *child nodes*

The algorithm traverses the hierarchy downwards until the spacing of the point samples is no larger than the minimum ray footprint. Then the point samples are intersected with the ray. Large triangles that are found on the way down in the hierarchy are tested, too. In the worst case, the algorithm terminates in the leaf nodes, where the remaining triangles are found. Finally, fragment compositing is done.

Mipmapping is done by traversing the hierarchy one step deeper after the matching resolution is found and then linearly blending together the result. Blending is performed automatically by the fragment compositing step; it suffices to provide the corresponding weights.

To calculate the minimum ray footprint (step 3), we perform again an eigenvalue computation (Equation 3) and determine the minimum diameter of the footprint ellipse. When the right level of detail is found, the

scene	#triangles unique / overall	triangle octree preproc.	triangle octree memory	point hierarchy preproc.	point hierarchy memory
two spheres	7 440 / 7 440	0.46 s	3.2 MB	5.7 s	2.3 MB
bunny-cow	110 231 / 8.7M	11.8 s	158 MB	443 s	106 MB

Table 1: Preprocessing time and storage costs

points and their intersection weights are inserted into the ray A-buffer. This leads to a footprint assembly [24] using prefiltered sample points with spacing just below the minimum diameter. The technique can lead to performance problems if the ray is very asymmetric because the bounding volumes of the octree as well as the sample point spacing are all symmetric. Therefore, we limit the anisotropy of the ray: If the minimum eigenvalue is smaller than a constant fraction (typically 1/4 - 1/16) of the maximum eigenvalue, we bound its value to this fraction of the maximum.

5.2 Stability and Efficiency

All intersection tests need an eigenvalue decomposition of the 2×2 ray footprint matrix (Equation 3), which is a quite expensive operation. The decomposition is needed to limit the spectral radii of the footprint as described above. Some tests like the triangle intersection test could theoretically be performed using the original footprint matrices. However, for the triangle test, this leads to wrong results in some cases due to numerical stability issues: If the two footprint coordinates are nearly colinear, the projection into footprint coordinates and evaluation of the edge-ray distances is numerically unstable. The eigenspace transformation allows us to perform the test using orthogonal coordinates avoiding the stability problems.

In order to reduce the number of expensive eigenvalue decompositions, one can perform the decomposition once per octree box and use the result for all intersection tests in the box. For our test scenes, usually no visible difference was observed in comparison to the exact version of the algorithm that performs the eigenspace transformation for every vertex using the exact footprint matrix at that point. This optimization was used for all example renderings and led to a speedup factor of about 2-3.

6 Results

We implemented a prototype of the multi-resolution raytracing algorithm in C++. The implementation has not been optimized for speed: We did not perform any cache or SIMD optimizations and only straightforward arithmetic optimizations. Therefore, the absolute speed cannot be compared with highly optimized raytracing packages such as the real-time raytracer of Wald et al. [26]. To allow a fair comparison of our technique with other raytracing techniques, we implemented a conventional raytracer based on the same code basis.

Special care was taken to implement both algorithms with a similar amount of optimizations. The conventional raytracer also uses an octree hierarchy (without points) and bounding sphere intersection tests for ray traversal. Auxiliary data structures such as mailboxes or geometric data structures use the same code in both implementations.

To compare the image quality with stochastic raytracing methods, we extended the conventional raytracer to an adaptive distribution raytracer. It uses two passes: In the first pass, an image with fixed oversampling is constructed (usually 1-3× oversampling, jittered rays). The resulting samples are used to estimate the image variance by examining the local neighborhood (usually 5×5 pixel). In the second pass, additional samples are placed in regions of high variance. The additional samples use stratified sampling: Each pixel is divided into a small grid and a jittered sample ray is shot in each grid cell. The elliptic area light sources are sampled uniformly with random ray positions. All tests were performed on a 2 GHz Pentium 4 with 1 GB of RAM.

We applied the different variants of the raytracing algorithms to different benchmark scenes. Table 1 summarizes the preprocessing costs. The rendering results are shown in Figure 6-8.

Low Complexity: The first scene (Figure 6) shows a reflective and a refractive sphere on a checker board, similar to the scene from the original cone tracing paper [3]. Our algorithm is able to render an antialiased solution that is nearly indistinguishable from the distributed raytracing result. The only difference that is slightly visible is the limitation of the ray anisotropy for the reflected rays that can be seen in the close-up of the red sphere. The overhead compared with non-antialiased standard raytracing is a factor of 4. The image contains a lot of sharp edges but only a few regions with highly uncorrelated noisy image content. Thus, the adaptive distributed raytracing is able to remove nearly all noise artifacts in the same time that is used by the multi-resolution algorithm.

High Complexity: The second benchmark scene is a model of higher complexity: Some well known meshes (the cow and the stanford bunny) are placed on a large checker board along with some procedural plants. The scene consists of about 8.7 million triangles[3] and the rendered images contain large areas of high variance. Especially the parts of the checker board and the plants that are farther away lead to high frequency details in the image. All elements of the scene, including the checker board, are modeled using geometry.

[3] The scene is described using hierarchical instantiation. The instantiated objects consist of 110231 unique triangles.

Therefore, techniques like texture prefiltering [12] cannot be used to remove noise artifacts.

The conventional raytracing solution shows severe aliasing artifacts (Figure 7). Our multi-resolution raytracer avoids most of the aliasing artifacts. The rendering time is again about 4 times larger than non-antialiased rendering. We rendered the same scene with the distribution raytracer and adjusted the oversampling parameters so that it used roughly the same amount of time as the multi-resolution raytracer. The resulting image still contains a considerable amount of noise. In contrast to the "low complexity" scene, the distribution raytracer is in this case not able to adapt the sampling density to save time as most of the image shows high frequency details. Sub-pixel stratification does not improve the convergence rate as it did in the first scene because the structure of the details is nearly random [18]. To obtain a noise free solution with stochastic sampling, we had to increase the rendering time by an order of magnitude (Figure 7d). This shows that the multi-resolution rendering algorithm provides an efficient technique for anti-aliased rendering of scenes containing highly detailed geometry. The algorithm delivers plausible images but there are still some issues:

The first issue is that the algorithm tends to overestimate the silhouettes of the objects. This is especially visible for the soft shadows and the silhouettes of the plant models. The problem is caused by the compositing step. The simple alpha blending with alpha values derived from the weight sums often overestimates the opacity of ray-surface intersections (although we already take into account the orientation and spacing of the sample points for the weight computation). A better solution would be to use subpixel masks for each ray (similar to the original A-buffer [6]) to estimate the mutual visibility of the fragments during compositing. This will be subject of future work.

A second issue is robustness: The base of the bunny in Figure 7b shows some holes. They appear because the compositing step merges the floor and the base of the bunny into a single fragment. Here, a refined criterion for fragment merging is needed. Some dark pixels at the back of the specular cow are caused by self-intersections of the outgoing ray cones with the reflecting surface. Currently, we use a threshold value for the minimum intersection distance based on the angle and the diameter of the outgoing ray cone. A more refined criterion would be necessary to avoid these artifacts.

Classic Cone Tracing: We also compare the runtime of the algorithm with classic cone tracing. For this purpose, we use our multi-resolution raytracer and disable the usage of point primitives. Thus, it is forced to intersect the ray cones with all triangles in the cone. For the low complexity scene, the running time was similar to the multi-resolution version. For the high complexity test scene the running time increases by a factor of 2.8 if we trace primary rays only. If we enable tracing of secondary rays, the performance of the purely triangle based cone tracer drops dramatically: The reflective and refractive secondary ray cones cover very large volumes in the scene and thus lead to an enormous amount of intersection tests. Therefore, we were not even able to measure a running time as the algorithm did not terminate within a reasonable amount of time. This shows that (recursive) cone tracing of complex scenes is not possible without a multi-resolution scene representation.

Special Effects: Figure 8 shows different special effects that can be implemented by altering the ray footprint. Image (a) shows approximate soft shadows. In comparison with the distributed raytracing solution (image (b)), the shadow region is again overestimated but the result image shows a plausible effect. Image (c) shows a depth of field effect. The three images in Figure 8d show blurry reflections with different roughness parameters. The roughness corresponds to an extra radial increment component in the *incrDev* ray parameter (see section 4.1). The special effects are not rendered as exactly as with distributed raytracing but appear to be plausible and do not show any noise artifacts. The main problem that remains is again the overestimation of opacity at the silhouette edges.

7 Conclusions

We presented a new approach for anti-aliased raytracing. The algorithm performs anisotropic cone tracing for a multi-resolution hierarchy of prefiltered points. In comparison to conventional raytracing, our algorithm is slower (by a factor of 4 for our implementation) but provides images without noise and aliasing artifacts. In contrast to classic cone tracing, the multi-resolution approach is able to deal with highly complex scenes for which the classic algorithm becomes prohibitively slow. In comparison with distributed raytracing [7], the new algorithm renders noise free solutions much faster, especially if the scene contains extended areas of unstructured content with high color variance. In such cases, the stochastic raytracer needs at least an order of magnitude more time to remove all noise artifacts. However, the image generated by the multi-resolution algorithm is only an approximation to the correct solution. The most important issue here is the overestimation of opacity at silhouette edges. In future work, we want to apply subpixel masks in the ray A-buffer to improve the accuracy of the silhouettes as well as that of soft shadows and depth-of-field effects.

Another direction for future work is an enhanced model for the ray surface interaction: Currently, the

point samples store only second order derivative information. Thus, it is not possible to capture reflective effects from noisy surfaces correctly (such as waves on the water seen from far away). We plan to include roughness information in the point samples, as proposed by Schilling [24] for the case of bump maps. Currently, the surface roughness is described by a material parameter.

Another direction is optimization for speed: The main reason for the larger rendering times of the multiresolution algorithm in comparison with conventional raytracing is the higher arithmetic complexity of the different intersection tests. Most of the arithmetic consists of linear transformations that can be mapped directly to SIMD instructions. The expensive square root operations in the eigenspace decompositions can be accelerated using lookup tables or iterative approximations. The hierarchy traversal could be accelerated by a better memory layout and more efficient bounding volumes and intersection tests [26].

Acknowledgements

The authors wish to thank Matthias Zwicker for valuable discussions concerning the fragment merging strategy, the anonymous reviewers for their helpful comments, and Michael Hauth for proofreading the paper.

References

[1] Adamson, A., Alexa, M.: Ray Tracing Point Set Surfaces. To appear in: *Shape Modelling International 2003*.

[2] Agrawala, M., Ramamoorthi, R., Moll, A.H.L.: Efficient Image-Based Methods for Rendering Soft Shadows. In: *SIGGRAPH 2000 Proceedings*, 375-384, 2000.

[3] Amanatides, J. Ray Tracing with Cones. In: *SIGGRAPH 84 Proceedings*, 18(3), 129-135, 1984.

[4] Amanatides, J., Object-Space Variance Estimators. In: *Proceedings of the Seventh Western Computer Graphics Symposium*, 85-87, 1996.

[5] Appel, A.: Some Techniques for Shading Mashine Renderings of Solids. In: *Proceedings of the Spring Joint Computer Conference*, 37-45, 1968.

[6] Carpenter, L.: The A-Buffer, an Antialiased Hidden Surface Method. In: *SIGGRAPH 84 Proceedings*, 103-108, 1984.

[7] Cook, R.L., Porter, T., Carpenter, L.: Distributed Raytracing. *SIGGRAPH 84 Proceedings,* 137-145, 1984.

[8] Genetti, J., Gordon, D, Williams, G.: Adaptive Supersampling in Object Space Using Pyramidal Rays. In: *Computer Graphics Forum, 17(1)*, 29-54, 1998.

[9] Glassner, A. S.: *Principles of Digital Image Synthesis.* Morgen Kaufmann Publishers, 1995.

[10] Ghanzanfarpour, D., Hasenfratz, J.M.: A Beam Tracing Method with Precise Antialiasing for Polyhedral Scenes. In: *Computer & Graphics, 22(1)*, 103-115, 1998.

[11] Heckbert, P., Hanrahan, P.: Beam Tracing Polygonal Objects. In: *SIGGRAPH 84 Proceedings*, 119-127, 1984.

[12] Igehy, H.: Tracing Ray Differentials. In: *SIGGRAPH 99 Proceedings*, 179-186, 1999.

[13] Jensen, H.W., Arvo, J., Fajardo, M., Hanrahan, P., Mitchel, D., Pharr, D., Shirley, P.: State of the Art in Monto Carlo Ray Tracing for Realistic Image Synthesis. In: *SIGGRAPH 2001 Course Notes*, Course 29, 2001.

[14] Kalaiah, A., Varshney, A.: Differential Point Rendering. In: *Rendering Techniques '01*, 2001.

[15] Keller, A: Quasi-Monte Carlo Radiosity. In: *Rendering Techniques '96*, 101–110. Springer, 1996.

[16] Kirk, D., The Simulation of Natural Features Using Cone Tracing. In: *The Visual Computer, 3(2)*, 63-71, Springer, 1987.

[17] Lischinski, D., Rappoport, A.: Image-based rendering for non-diffuse synthetic scenes. In: *Rendering Techniques '98*, 301-314, 1998.

[18] Mitchell, D.P.: Consequences Of Stratified Sampling In Graphics. In: *SIGGRAPH 96 Proceedings*, 354-376, 1996.

[19] Mitchell, D.P.: Generating Antialiased Images at Low Sampling Densities. In: *SIGGRAPH 87 Proceedings*, 65-72, 1987.

[20] Painter, J., Sloan, K.: Antialiased ray tracing by adaptive progressive refinement. In: *SIGGRAPH 89 Proceedings*, 23(3), 281-288, 1989.

[21] Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: Surface Elements as Rendering Primitives. In: *SIGGRAPH 2000 Proceedings*, 335-342, 2000.

[22] Rusinkiewicz, S., Levoy, M.: Qsplat: A Multiresolution Point Rendering System for Large Meshes. In: *SIGGRAPH 2000 Proceedings*, 343-352, 2000.

[23] Schaufler, G., Jensen, H.W.: Ray tracing point sampled geometry, *Rendering Techniques 2000*, 319-328, Springer, 2000.

[24] Schilling, A.: Antialiasing of Environment-Maps, In: *Computer Graphics Forum, 20(1)*, 5-11, 2001.

[25] Shinya, M., Takahashi, T., Naito, S.: Principles and applications of pencil tracing. In: *SIGGRAPH 87 Proceedings*, 45-54, 1987

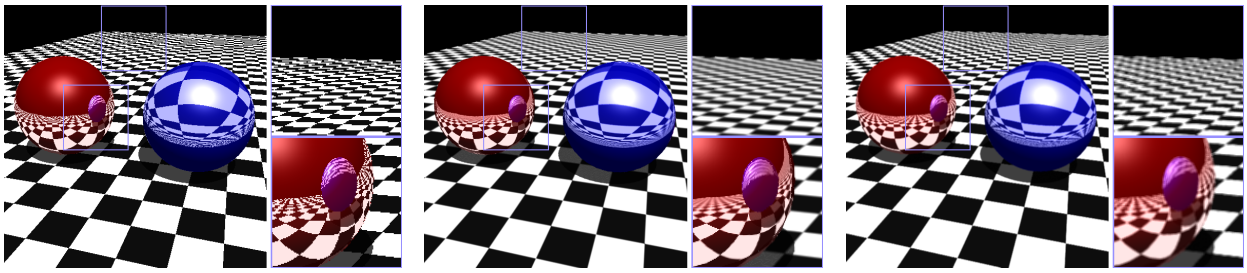
[26] Wald, I., Slusallek, P., Benthin, C., Wagner, M.: Interactive Rendering with Coherent Raytracing. In: *Computer Graphics Forum, 20(3)*, 153-164, 2001.

[27] Wand, M., Fischer, M., Peter, I., Meyer auf der Heide, F., Straßer, W.: The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes, In: *SIGGRAPH 2001 Proceedings*, 361-370, 2001.

[28] Wand, M., Straßer, W.: Multi-Resolution Rendering of Complex Animated Scenes. In: *Computer Graphics Forum, 21(3)*, 483-491, 2002.

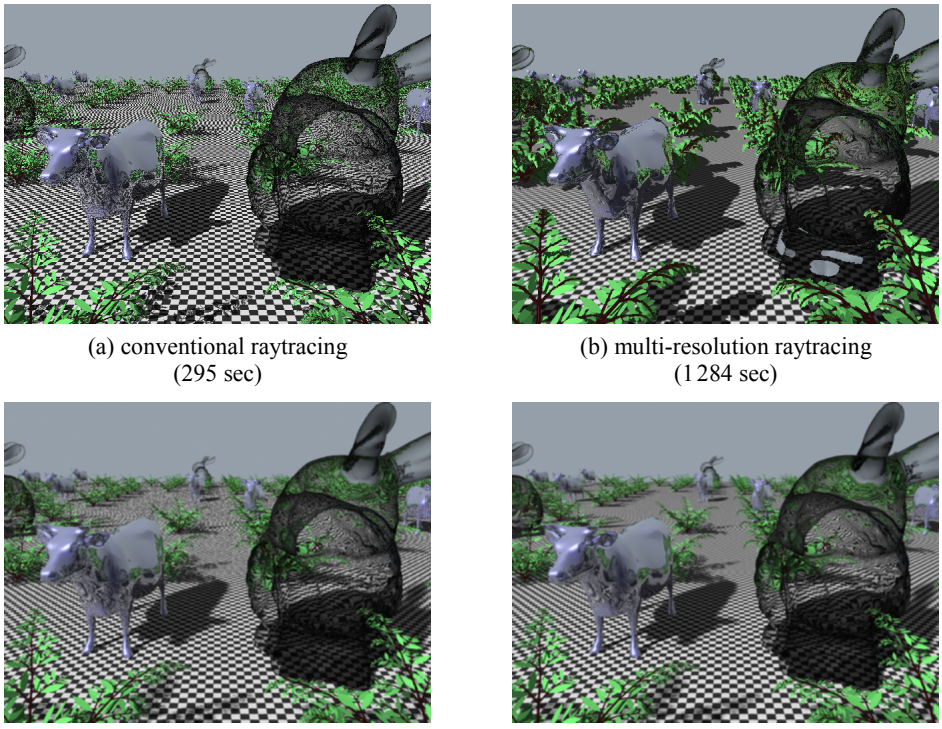
[29] Whitted, T.: An Improved Illumination Model for Shaded Display, *Communications of the ACM*, 23(6), 343-349, 1980.

[30] Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface Splatting. In: *SIGGRAPH 2001 Proceedings*, 371-378, 2001.



(a) conventional raytracing (25 sec) (b) multi-resolution raytracing (99 sec) (c) distributed raytracing (99 sec)

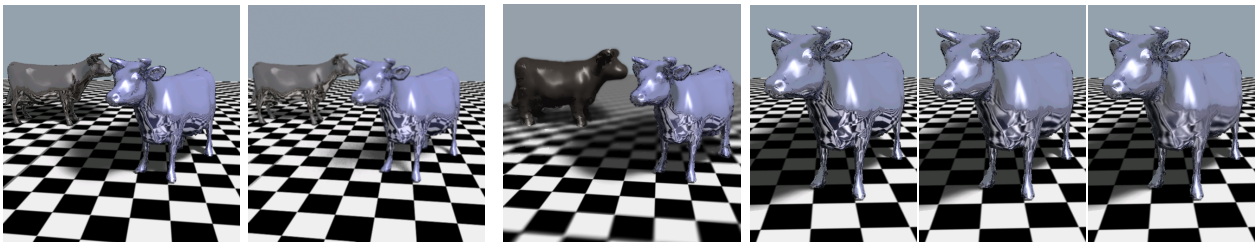
Figure 6: Comparison between conventional, multi-resolution, and distributed raytracing (512 × 512 pixel). Note that the scene consists of triangles only (the checker board is geometry, not a texture).



(a) conventional raytracing (295 sec) (b) multi-resolution raytracing (1 284 sec)

(c) distributed raytracing (1 444 sec) (d) reference solution: distributed raytracing with large oversampling (15 412 sec)

Figure 7: A more complex scene: "Specular Cows, Transparent Bunnies and Plants" (640 × 480 pixel). The scene consists of 8.7 million triangles, modeled using scene graph based hierarchical instantiation.



a) soft shadows (multi-resolution raytracing) b) soft shadows (distributed raytracing) c) depth of field d) blurry reflections (increasing surface roughness from left to right)

Figure 8: Special Effects (512 × 512 pixel)